C++ Style Guidelines

The following are guidelines for programming in C++. The guidelines must be followed in class to get your submissions accepted for full credit.

Naming Conventions

- All variables, functions, classes and structure names use camelCase.
- Never use an underscore in a name.
- Variables always start with a lower case first letter.
- Function names always start with a lower case first letter.
- Structures+Classes always start with an upper case first letter.
- Globals and constants are always all upper case with no lower case letters.

Commenting

Use // for short comments and `/*` and `*/` for long comments or blocking out code.

Well written code should need very few comments. Use good function names and good variables names and avoid comments as much as possible. Comments should be written where they are needed to explain confusing code. If you do not write confusing code, then you should not have many comments.

There are a few required peices of documentation your code must have. These are Doxygen Comments.

1) The program must have a description in main.cpp
2) Each file must have a description (in main this is in addition to the program description).
3) Each function needs a comment above the prototype.
4) Each structure+class needs a comment at the definition.

The Doxygen comments use two * at the start `/**` instead of `/*` for regular comments.

Program Documentation

This is placed only in main.cpp

```
/**
 @mainpage CS ### - Homework #
 @section Description

 Describe how the entire program works. What is it for?

 */
```

File Documentation

This is placed in every file. In `main.cpp` it comes after the program description.

```
/**
  @file
  @author Mark Boady <mwb33@drexel.edu>
  @date July 20, 2022
  @section DESCRIPTION

  Write a description of what this file contains here.
 */
```

Function Documentation

Each function must have a documentation block. The `@return` and `@param` values are optional. Only put them in if you have return values or parameters. This examples uses both. You may have as many `@param` commands as needed.

These are always placed with the prototype.

```
/**
 Draw a right triangle using stars.
 @param x is the number of lines to draw.
 @return 0 for sucess and 1 for failure
*/
int triangle(int x);
```

Structure/Class Documentation

A structure has a comment above it with a description of the structure. Each value stored in the structure also has a comment.

Structure Example:

```
/**
  A structure to represent a heap (Priority Queue / Min Heap) Data Structure.
 */
typedef struct Heap Heap;
struct Heap{
  int* data;/**< A pointer to your array of numbers. */
  int maxSize;/**< The maximum size of the heap before it needs to be resized.*/
  int currentSize;/**< The current number of items in the array. */
};
```

Class Example:

```
/**
 This class represents one pixel in an image.
 */
```

```
class pixel{
private:
    unsigned char red;/**< The value of the red component of the pixel.*/
    unsigned char green;/**< The value of the green component of the pixel.*/
    unsigned char blue;/**< The value of the blue component of the pixel.*/
    // Methods are commented just like functions above!
public:
    /**
        The default constructor creates a black pixel.
     */
    pixel(){
        red = 0;
        green = 0;
        blue = 0;
    }
```

Citing Sources

When you want to cite a source, note the begininng and end of the referenced code. You do not need to cite the course website, slides etc. Any material provided by the Professor does not need a citation.

```
/* Start of Cited Code
   Describe Reference (website, person, ect): ?
   Provide a link for any website: ?
*/
for(int i=0; i< 10; i++){
    std::cout << i << std::endl;
}
/* End of Cited Code */
```

Code Blocks

When writing code blocks the starting bracket is on the same line as the keywork/function.

There should be no spaces between keywords and parenthesis.

Examples:

```
int f(int x){
  int total =0;
  for(int i=0; i < x; i++){
    if(i % 2 == 0){
      total = total + 2;
    }else{
      total = total * 2;
    }
  }
  return total;
}
```

Code Organization

- All functions must have prototypes.
- Break Classes and Complicates code into
  - filename.h the class and function defs
  - filename.cpp the class and function implementation

Other Guidelines

- Never use `goto` commands.
- Programs always end with a newline.
- Always favor classes over structures.
- Avoid globals when possible. If you use a global, provide a comment explaining why it was needed.
- Operators look better have spaces around them `9 + 2` instead of `9+2`.